04101 Abstracts Collection
Language Engineering for Model-Driven Software Development
— Dagstuhl Seminar —

J. Bézivin[1] and R. Heckel[2]

[1] University of Nantes, France
Jean.Bezivin@sciences.univ-nantes.fr
[2] Universität Dortmund, Germany (on leave from Paderborn)
reiko@upd.de

Abstract. From 29.02. to 05.03.04, the Dagstuhl Seminar 04101 "Language Engineering for Model-Driven Software Development" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

04101 Summary – Language Engineering for Model-driven Software Development

Jean Bézivin, Reiko Heckel

This paper summarizes the objectives and structure of a seminar with the same title, held from February 29th to March 5th 2004 at Schloss Dagstuhl, Germany.

Keywords: Dagstuhl Seminar 04101
Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/10

04101 Discussion – A Taxonomy of Model Transformations

Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp

This report summarises the results of the discussions of a working group on model transformation of the Dagstuhl Seminar on Language Engineering for Model-Driven Software Development. The main contribution is a taxonomy of model transformation. This taxonomy can be used to help developers in deciding which model transformation approach is best suited to deal with a particular problem.

Keywords: Taxonomy, model transformations

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/11

Models as first class entities

Jean Bézivin (Université de Nantes)

In November 2000, the OMG made public the MDA^{TM} initiative, a particular variant of a new global trend called model engineering. The basic ideas of model engineering are germane to many other approaches such as generative programming, domain specific languages, model-integrated computing, software factories, etc. MDA may be defined as the realization of model engineering principles around a set of OMG standards like MOF, XMI, OCL, UML, CWM, SPEM, etc. Similarly to the basic principle "Everything is an object" that was important in the 80's to set up the object-oriented technology, we suggest, in model engineering, that the basic principle "Everything is a model" may be key to identifying the essential characteristics of this new trend. To assess this principle, we need to characterize the two associated relations, nameley the relations of "representation" and "conformance".

Keywords: MDE, MDA, Model Transformation

See also: Novatica Journal, Special Issue, March-April 2004

Application of Graph Transformation for Automating Web Service Discovery

Alexey Cherchago (Universität Paderborn)

The paper represents current achievements of an ongoing research that aims to develop a formal approach supporting an automatic selection of a Web service sought by a requestor. The approach is based on the matching the requesto's requirements for a "useful" service against the service description offered by the provider. We focus on the checking behavioral compatibility between operation contracts specifying pre-conditions and effects of required and provided operations. Graph transformation rules with positive application conditions are proposed as a visual formal notation for contracts. The desired dependence between requestor and provider contracts is determined by the semantic compatibility relation and syntactic matching procedure that is sound w.r.t. this relation.

Keywords: SOA, graph transformation, contracts

Joint work of: Heckel, Reiko; Cherchago, Alexey

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/12

Generative Software Development

Krzysztof Czarnecki (University of Waterloo)

System family engineering seeks to exploit the commonalities among systems from a given problem domain while managing the variabilities among them in a systematic way. In system family engineering, new system variants can be rapidly created based on a set of reusable assets (such as a common architecture, components, models, etc.).

Generative software development aims at modeling and implementing system families in such a way that a given system can be automatically generated from a specification written in a textual or graphical domain-specific language.

In this talk, I will give an overview of the generative development process, including domain analysis (i.e., capturing the commonalities and variabilities within a system family using feature modeling), domain design (i.e., developing a common architecture for a system family), and implementating program generators using different technologies, such as template-based code generation, C++ template metaprogramming, and model transformations. Finally, I will demonstrate tool support for feature modeling and discuss the relationship of generative software development to Model Driven Architecture.

MDA is Language Design and Translation

Keith Duddy (DSTC - Brisbane)

Since the first profiles for UML were defined, and the first plugins written to generate code skeletons from UML models, the modelling trick of marking up standard models, and the programming trick of traversing those models and outputting code fragments have merged into the paradigm of domain language design and translation. This fact is being realised by OMG standards: The UML is now aligned with the MOF modelling language design standard of OMG, and a new standard for model transformation (a.k.a language translation) is being developed by the leading researchers and tool vendors. This talk constrasts the promise of modelling tools becoming rich a language design and translation toolkit against the danger of allowing the Model Driven Architecture (TM) to be limited to attaching strings to class models which direct opaque code generators.

Keywords: Language design, MOF, OMG, model transformation, language translation, domain languages

Foundations of Model (Driven) (Reverse) Engineering : Models
Episode I: Stories of The Fidus Papyrus and of The Solarus

Jean-Marie Favre (LSR - IMAG)

Model Driven Engineering (MDE) received a lot of attention in the last years, both from academia and industry. However, there is still a debate on which basic concepts form the foundation of MDE. The Model Driven Architecture (MDA) from the OMG does not provided clear answers to this question. This standard instead provides a complex set of interdependent technologies. This paper is the first of a series aiming at defining the foundations of MDE independently from a particular technology. A megamodel is introduced in this paper and incrementally refined in further papers from the series. This paper is devoted to a single concept, the concept of model, and to a single relation, the RepresentationOf relation. The lack of strong foundations for the MDA 4-layers meta-pyramid leads to a common mockery: "So, MDA is just about Egyptology?!". This paper is the pilot of the series called "From Ancient Egypt to Model Driven Engineering". The various episodes of this series show that Egyptology is actually a good model to study MDE.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/13

Foundations of Meta-Pyramids: Languages vs. Metamodels
Episode II: Story of Thotus the Baboon

Jean-Marie Favre (LSR - IMAG)

Despite the recent interest for Model Driven Engineering approaches, the so-called four-layers metamodelling architecture is subject to a lot of debate. The relationship that exists between a model and a metamodel is often called instanceOf, but this terminology, which comes directly from the object oriented technology, is not appropriate for the modelling of similar meta-pyramids in other domains. The goal of this paper is to study which are the foundations of the meta-pyramids independently from a particular technology. This paper is actually the second episode of the series "From Ancient Egypt to Model Driven Engineering". In the pilot episode, the notion of megamodel was introduced to model essential Model Driven Engineering concepts. The notion of models was thoroughly discussed and only one association, namely RepresentationOf was introduced. In this paper the megamodel is extended with one fundamental relation in order to model the notion of languages and of metamodels. It is shown how Thotus the Baboon helped Nivizeb the priest in designing strong foundations for meta-pyramids. The secrets of some ancient pyramids are revealed.

Keywords: Models, reverse engineering, transformations

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/21

How far can you push UML profiles?

Tracy Gardner (IBM Winchester)

UML profiles are the built-in customization mechanism for UML. A profile is a subset and extension of UML that is designed for a specific task or domain. This approach can be used to design new modeling languages. This talk will discuss how far we can push the notion of UML profiles. We will cover both the limits of UML profiles as they stand today (and in the upcoming UML 2.0) and what could be done to extend the concept still further.

As a case study we use the UML Profile for Automated Business Processes which has a mapping to the Business Process Execution Language for Web Services. This work is currently being extended as part of a response to the OMG's Business Process Definition Metamodel RFP to support business level users in additionl to the IT architects and developers supported by the earlier profile. The BPD Metamodel being proposed is a profile of UML 2.0 with first-class extensions only for small parts of the metamodel where there is no good fit in UML 2.0.

(An Example for) Metamodeling Syntax and Semantics of Two Languages, their Transformation, and a Correctness Criterion

Martin Gogolla (Universität Bremen)

We study a metamodel for the Entity Relationship (ER) and the Relational data model. We do this by describing the syntax of the ER data model by introducing classes for ER schemata, entities, and relationships. We also describe the semantics of the ER data model by introducing classes for ER states, instances, and links. The connection between syntax and semantics is established by associations explaining that syntactical objects are interpreted by corresponding semantical objects. Analogously we do this for the Relational data model. Finally, we give a metamodel for the transformation of ER schemata into Relational database schemata. By characterizing the syntax and semantics of the languages to be transformed and also the transformation itself within the same (meta-)modeling language we are able to include equivalence criteria on the syntactical and on the semantical level for the transformation. In particular, we show that the semantical equivalence criterion requires that the ER states and the corresponding Relational states bear the same information.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/14

A modelbased development process for automotive electronic systems

Ursula Goltz (TU Braunschweig)

In the automotive area the majority of today's innovations results from an increasing portion of software functionality, which cannot be mastered any more by conventional development concepts. The growing complexity of such systems therefore requires new methods for the development of embedded systems.

Goal of the project STEP-X is to propose a seamless development process starting with the definition of requirements and leading to an automatic code-generation according to the V-Modell. Due to the extended significance of quality assurance and system reliability the topics test and diagnosis are strongly integrated into the model based process. An important constraint is that only commercially available tools are evaluated and introduced, in order to ensure further use of the projec's results.

In the talk, the models used in the STEP-X method and the occurring problems will be discussed.

Behaviour Behaviour and Language Engineering

Luuk P.J. Groenewegen (Leiden University)

Language engineering, among other things, aims at providing support for changing programming and modelling languages, such that a program (or model) can change into a next program (or model), possibly written in a different language, possibly while in execution, i.e. on-the-fly.

In the talk we take the following position: Paradigm's behaviour behaviour notions provide a dynamic structuring relevant for building such changes on.

To that aim, we present behaviour behaviour (global behaviour) in terms of the Paradigm notions of subprocess and trap. In addition we sketch how these notions actually unify modelling of coordination, of evolution on-the-fly and of mobility. Finally we argue why these notions could be relevant for the kind of problems language engineering wants to address.

Taking the position thus defended, we propose to develop notions for (other) modelling languages and programming languages, similar to the Paradigm notions of subprocess and trap. This then should add a new kind of behavioural structuring to these languages, thus facilitating both specification and control of behaviour behaviour within models or programs written in these languages. This in turn could provide a new basis within such languages for supporting the kind of changes language engineering addresses.

Keywords: Behaviour behaviour, Paradigm, evolution on-the-fly, JIT modelling, JIT informing, language engineering

Language Engineering in Practice

Martin Große-Rhode (FhG - ISST Berlin)

The Dependable Systems Department of the Fraunhofer ISST Berlin creates methods for the development, integration, and maintenance of embedded automotive systems. Based on a thourough analysis of the existing processes, domain- and enterprise-specific roles, activities, and artefacts are designed for an immediate enhancement of the technical, processual, and organsiational infrastructure that is found at the company. General principles that govern the design of the methods are continuous model-based engineering and domain engineering. Continuous model-based engineering first means to focus the development process on models whose structure is designed in such a way as to optimally support the activities and roles found in the foregoing analysis. Second, these models are interconnected via a common core, such that the transitions between the activities are also fully reflected at the model level. Thereby design decisions can be traced throughout the whole development process, consistency can be checked, and changes can be managed. Domain engineering aims at a systematic reuse by the generation of models or model templates as analysis, design, and implementation assets and techniques for their reuse within the development of new products.

As a means to define the abstract modelling language that determines the structure of the models that are to be used a two-step meta-modelling ap- proach turned out as most adequate. In the first step class diagrams are used to introduce the modelling elements and their fundamental relationships. Since just classes, binary associations with multiplicities, and attributes are used any object-oriented modelling language or tool can be used for that. To make the meta-model complete constraints have to be added that define the relation- ships of the modelling elements more precisely. In principle any logic or ob- ject constraint language can be used for that purpose. We have chosen the object-oriented extension ObjectZ of the set-theoretic specification language Z, although it implied to reformulate the whole class diagram developed before within the constraint set. This decision was based essentially on the clarity of the language, the time constraints of the projects, and the previous knowledge of the team members.

Within the development and integration process the following main views are distinguished and supported by appropriate models: requirements, logical architecture, and technical architecture including hardware and software architecture. As mentioned above, also the interconnections of the views are specified by models. That means that there are realisation models that connect different requirements models (like user requirements, legal requirements, or system requirements) among each other and with the other models (logical and technical architecture), and that there are models for the partitioning of the logical components onto the technical (software and hardware) components.

Flexibility in the design of the modelling languages (and thus the models) is achieved by a layered meta-modelling approach. Thereby the modelling elements are introduced within a hierarchy, with most general elements at the top and

stepwise refinements to introduce more domain-, enterprise-, and role-specific concepts. Concerning requirements models for example at the top layer the general structure of requriements is defined: a requrirement consists of a subject or stake holder (the one who requires the feature), a feature that is required, possibly a mode (the feature must or may be included), and an object or target, i.e. the system or the group of systems under development that shall have this feature. These constituents can than be refined, for example by saying which groups of stake holders are relevant, which kinds of featuers are to be distinguished (functional, non-functional, safety, etc.), which modes are there, which kinds of groups of systems are considered and how are they described. Analoguously the modelling concepts for the other views and the modelling concepts for the domain models that capture the reusable analysis, design, and implementation assets are introduced. The experience of our projects with industrial partners shows that this kind of language engineering is an adequate means for the introduction of continuous model-based software development processes in the industrial practice.

Technical reports on method developments including the correspondingmeta-models will be made available soon via the web page of the author.

Keywords: Model-based systems engineering, embedded automotive systems, requirements; architecture, language engineering

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2005/15

Languages for Automated Model Based Software Testing

Alan Hartman (IBM - Haifa)

The AGEDIS project [1] was a three year endeavor by a consortium of industrial and academic partners to create an infrastructure for model based testing. In the course of this project five languages were defined. The purpose of these languages are a) for modeling software [2], b) for describing test objectives [2,3], c) for describing testing artifacts (test suites, and test trace logs) [4], d) for describing the simulation interface to the model [3], and e) for describing the testing architecture [5]. This paper discusses these five languages and the design decisions that were taken during the course of the project. We attempt to draw conclusions with hindsight, and make some proposals for the improvement and standardization of some of these languages.

References:

1. AGEDIS Consortium, Final Report, http://www.agedis.de
2. AGEDIS Consortium, AGEDIS modeling language specification, http://www.agedis.de
3. AGEDIS Consortium, Intermediate Language 2.0 with Test Directives Specification, http://www.agedis.de
4. AGEDIS Consortium, Test Suite Specification, http://www.agedis.de
5. AGEDIS Consortium, TED Users Guide, http://www.agedis.de

UML semantics - Dynamic Meta Modeling and beyond

Jan Hendrik Hausmann (Universität Paderborn)

While the notations and concepts of the UML are constantly refined, the semantics of this important language is still officially provided in prose only. Over the time a number of established techniques for defining language semantics have been applied to UML. Each of these approaches improved the precision, but none can be considered as broadly accepted.

In our work we try to supply a semantics description to UML which adheres to the same goals as UML itself, namely being understandable by engineers (this includes being diagrammatic), bringing benefits to a broad user group (rather than please some specialists), being extendable/modular and being tool-independent. To be superior to the current style of definition, it also needs to be formal and precise.

From the requirements we developed the technique of Dynamic Meta Modeling [EHHS00], which was originally influenced by Plotkin's Structured Operational Semantics [Plo81,CHM00] but is currently being adapted to ideas present in Mosses' Action Semantic [Mos96]. This essentially means that we define a semantics domain by means of meta modeling, provide a denotation mapping [HK03] into this domain and specify graph transformation rules which provide operational semantics for this domain.

In this talk we will report on the technique of Dynamic Meta Modeling and our experiences in applying it. We found a number of examples which work well (and which we are happy to show) and some pitfalls where special cases threatened to destroy some properties of our semantics. Because DMM is a very powerful and flexible technology, defining a semantics by DMM requires a lot of decisions, each of which might include a compromise between different requirements.

References:

[CHM00] A. Corradini, R. Heckel, and U. Montanari. Graphical operational semantics. In Proc. ICALP2000 Workshop on Graph Transformation and Visual Modelling Techniques. Carleton Scientific, 2000. 2000/CorradiniGTVMT00.pdf.

[EHHS00] G. Engels, J.H. Hausmann, R. Heckel, and St. Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In A. Evans, S. Kent, and B. Selic, editors, Proc. UML 2000, York, UK, volume 1939, pages 323-337. Springer-Verlag, 2000.

[HK03] J.H. Hausmann and S. Kent. Visualizing model mappings in UML. In Proc. of the ACM Symposium on Software Visualization 2003, 2003. to appear.

[Mos96] Peter D. Mosses. Theory and practice of action semantics. In MFCS '96, Proc. 21st Int. Symp. on Mathematical Foundations of Computer Science (Cracow, Poland, Sept. 1996), volume 1113, pages 37-61. Springer-Verlag, 1996.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.

Keywords: UML, meta modeling, denotational semantics, operational semantics, graph transformation

See also: J. H. Hausmann, R. Heckel, S. Sauer; Dynamic Meta Modeling with Time: Specifying the Semantics of Multimedia Sequence Diagrams; Journal on Software and Systems Modeling, Volume 3, Number 3. Springer August 2004, pages 181-193.

Model-based Development of Web Services: A crash course

Reiko Heckel (Universität Paderborn)

The talk is intended as an introduction to the case study proposed by the organizers. I will cover the main concepts of Web services languages and protocols, as well as their modeling in UML.

The talk will also identify language design concerns arising from the example, that may be addressed by the techniques and tools of the participants.

See also:
http://www.upb.de/cs/ag-engels/Conferences/Dagstuhl04101/html/example.html

Graph Transformation in a Nutshell

Reiko Heckel (Universität Paderborn)

Even sophisticated techniques start out from simple ideas. Later, in reply to application needs or theoretical problems new concepts are introduced and new formalizations proposed, often to a point where the original simple core is hardly recognizable. In this paper we provide a non-technical introduction to the basic concepts of typed graph transformation systems, completed with a survey of more advanced concepts, and explain some of its history and motivations.

Keywords: Graph transformation

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/16

A MDA Approach to Model & Implement Transformations

Jean-Marc Jézéquel (IRISA - Rennes)

Only in software and in linguistics a model has the same nature as the thing it models. In software at least, this opens the possibility to automatically derive software from its model. This property is well known from any compiler writer (and others), but it was recently be made quite popular with an OMG initiative called the Model Driven Architecture (MDA). The model transformations allowing the engineers to more or less automatically go from platform-independent models (PIM) to platform-specific models (PSM) are increasingly seen as vital assets that must be managed with sound software engineering principles. We believe that transformations should be first-class models in the MDA world; we propose to adopt the object-oriented approach and to leverage the expressive power of UML as a metamodel defining the transformation language.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/20

Model Transformation

Frédéric Jouault (Université de Nantes)

The ATLAS group (INRIA & LINA) is currently involved in porting a model transformation language named ATL to Eclipse under the GMT project. This paper describes the main characteristics of model transformation environments, of the current version of ATL and of the planned next version intended to be integrated into Eclipse. The demonstration will be built on a very simple example, but references to other non-trivial real life examples will be made in the paper.

Multi-Domain Integration with MOF and extended Triple Graph Grammars

Alexander Königs (TU Darmstadt)

One aim of tool integration is designing an integrated development environment that accesses the data/models of different tools and keeps them consistent throughout a project being considered. Present approaches that aim for data integration by specifying (graphically denoted) consistency checking constraints or consistency preserving transformations are restricted to pairs of documents. We present an example that motivates the need for a more general data/model integration approach which is able to integrate an arbitrary number of MOF-compliant models. From a formal point of view this approach is a generalization of the triple graph grammar document integration approach. From a practical point of view it is a proposal how to specify multidirectional declarative model transformations in the context of OMG's model-driven architecture (MDA) development efforts and its request for proposals for a MOF-compliant "query, view, and transformation" (QVT) approach.

Joint work of: Alexander Königs, Andy Schürr

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/22

Deep Characterization

Thomas Kühne (TU Darmstadt)

More and more multi-level description hierarchies are employed in various areas. For instance, in the definition of modeling languages, process modeling, and domain models with a dynamic type level. In each of these applications at least three description levels are used which are related to each other by the "instance-of" relationship. However, traditional instantiation semantics is designed to relate two levels only and thus fails to support the sometimes desired ability to classify across more then one level boundary. The concept of "deep instantiation"–a conservative extension of two-level instantiation–is proposed to support the characterization of model elements across multiple model boundaries. Some of the implications for model driven development are discussed and a few questions are raised as to what exactly deep instantiation should support and what not.

Keywords: Metamodeling, instantiation, powertype

What is a Model?

Thomas Kühne (TU Darmstadt)

With the recent trend to model driven development a commonly agreed notion of "model" becomes a pivotal issue. However, currently there is little consensus about what exactly a model is and what it is not. Furthermore, basic terms such as "metamodel" are far from being understood in the same way by all members of the modeling community. This article attempts to start establishing a consensus about generally acceptable terminology. Its main contribution is the distinction between two fundamentally different kinds of models, i.e. "type model" versus "token model". The recognition of the fundamental difference in these two kinds of models is crucial to avoid misunderstandings and unnecessary disputes among members of the modeling community.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/23

Model Transformations in the Consistency Workbench

Jochen Küster (Universität Paderborn)

For the purpose of behavioral consistency checking, translations of UML models into arbitrary formal languages are required, depending on the kind of consistency problem being treated. The Consistency Workbench aims at supporting the software engineer in developing such model transformations and consistency checks on the basis of the defined model transformations. The concept of our model transformation approach is based on graph transformation where a so-called compound rule describes the transformation of both source and target model. In this talk, we first explain this concept of compound rules and illustrate their implementation in the Consistency Workbench. Then, we elaborate on the validation of model transformations and briefly describe sufficient criteria for ensuring the confluence of model transformations.

Keywords: Consistency, model transformation

Towards an engineering discipline for GRAMMARWARE

Ralf Lämmel (Vrije Universiteit Amsterdam)

Grammarware comprises grammars and all grammar-dependent software, i.e., software that directly involve grammar knowledge. The term grammar is meant here in the widest sense to include XML schemas, syntax definitions, interface descriptions, APIs, and interaction protocols. Typical examples of grammar-dependent software are document processors, parsers, import/export functionality, and generative programming tools. Despite this pervasive role of grammarware in software systems, it is somewhat neglected — from an engineering point of view. We lay out an agenda that is meant to promote research on improving the quality of grammarware and on increasing the productivity of grammarware development. To this end, we identify the problems with current foundations and practices, the promises of an engineering discipline for grammarware, its ingredients, and research challenges along the way.

Keywords: Grammarware

Joint work of: Lämmel, Ralf; Klint, Paul; Verhoef, Chris

Full Paper: http://www.cs.vu.nl/grammarware/

See also: P. Klint, R. Lämmel, and C. Verhoef. Towards an engineering discipline for grammarware. Under revision for the ACM Transactions on Software Engineering and Methodology

UML Model Evolution and Inconsistency Management

Tom Mens (Université de Mons)

A software design is often modelled as a collection of UML diagrams. There is an inherent need to preserve their consistency, since these diagrams are subject to continuous changes due to successive refinements or evolutions. Contemporary UML tools provide unsatisfactory support for maintaining the consistency between different versions of UML diagrams. To solve this problem, an extension of the UML metamodel is developed, and a classification of inconsistencies is proposed. Detection and resolution of inconsistencies is expressed by means of rules in description logic. By carrying out a number of concrete experiments, we show the feasibility of the description logic formalism for the purpose of managing inconsistencies between evolving UML models. We also show how we can use the formalism and tool to propose model refactorings that improve the design.

Keywords: Consistency maintenance, inconsistency management, UML, description logic, model evolution, design improvement, model refactoring, XML

Joint work of: Mens, Tom; Van Der Straeten, Ragnhild

Action semantics

Peter D. Mosses (BRICS - Aarhus)

Action Semantics (developed together with David Watt in the late 1980's [2, 3, 5, 6]) is a hybrid of denotational and operational semantics. It allows the semantics of individual (programming language) constructs to be specified incrementally, i.e., separately and independently [1].

In action semantics, the denotations of constructs (modelling their contributions to overall behaviour) are specified in action notation, which is itself defined operationally [2, 4]. Action notation provides primitive actions and combinators for expressing fundamental programming concepts, including the flow of control and data, binding, storage, and asynchronous communication between separate processes.

Although action notation has some features in common with the way that actions and activities are specified in UML 2.0,[1] there are also significant differences. The action notation used in action semantics aims at simplicity, economy, and algebraic elegance, and it is not tailored to the description of object-oriented (or other particular kinds of) systems.

References:

1. K.-G. Doh and P. D. Mosses. Composing programming languages by combining action-semantics modules. Sci. Comput. Programming, 47(1):3-36, 2003.
2. P. D. Mosses. Action Semantics. Number 26 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

3. P. D. Mosses. Theory and practice of action semantics. In MFCS '96, Proc. 21st Int. Symp. on Mathematical Foundations of Computer Science (Cracow, Poland, Sept. 1996), volume 1113 of LNCS, pages 37-61. Springer-Verlag, 1996.
4. P. D. Mosses. A modular SOS for Action Notation (extended abstract). In AS'99, number NS-99-3 in BRICS Notes Series, pages 131-142, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1999. Full version available at http://www.brics.dk/RS/99/56/.
5. P. D. Mosses and D. A. Watt. The use of action semantics. In Formal Description of Programming Concepts III, Proc. IFIP TC2 Working Conference, Gl. Avernæs, 1986, pages 135-166. North-Holland, 1987.
6. D. A. Watt. Programming Language Syntax and Semantics. Prentice-Hall, 1991.

[1] http://www.uml.org

Model-Driven Development for the Web

Pierre-Alain Muller (ESSAIM - Mulhouse)

Model engineering is the opportunity to re-inject software engineering in the development of web applications.

We have developed metamodels and action languages to represent web applications via three aspects (business, hypertext and presentation).

Our goal is to allow the complete modeling of web application in the MDA TS. We are currently able to build web PIMs and then translate them into web PSMs (Java, PHP, Oracle, MySQL, HTML, DHTML...).

Keywords: Model-driven engineering, MDA, web, matamodel, PIM

A Modeling Language with operational, but not necessarily executable semantics

Wolfgang Reisig (HU Berlin)

After many years of research and exercise towards adequate modeling languages, the question arises as to whether there exists a theoretical foundation of "specification", in analogy to the computable functions, which provide the foundation of programming.

Conventional semantics is based on states s: var ? val, where var and val are sets of variables and values, respectively. A run is a finite or infinite sequence of states, and a specification is a finite description of a set of runs. Conventional semantics is based on the assumption that each value is a finite or infinite sequence of symbols. The specification is implementable in this case.

We are interested in the case where values are any items. Examples are the basic objects of midleware programs. Toy examples are geometrical algorithms with points, circles, lines etc as basic elements, and operations such as the construction of a line from two nodes, or the intersection points of intersecting circles.

An adequate description of a specification of this kind is based on a signature ?; viz. a collection of operation symbols, each with its arity. A specification is then essentially a ?-term, and a state is a ?-structure. This is in fact the basics of Gurevich's Abstract state machine approach. This proposal may eventually be an adequate starting point for a theory of specification.

Subjects, Models, Languages, Transformations

Arend Rensink (University of Twente)

Discussions about model-driven approaches tend to be hampered by terminological confusion. This is at least partially caused by a lack of formal precision in defining the basic concepts, including that of "model" and "thing being modelled" which we call subject in this paper. We propose a minimal criterion that a model should fulfill: essentially, it should come equipped with a clear and unambiguous membership test; in other words, a notion of which subjects it models. We then go on to discuss a certain class of models of models that we call languages, which apart from defining their own membership test also determine membership of their members. Finally, we introduce transformations on each of these layers: a subject transformation is essentially a pair of subjects, a model transformation is both a pair of models and a model of pairs (namely, subject transformations), and a language transformation is both a pair of languages and a language of model transformations. We argue that our framework has the benefits of formal precision (there can be no doubt about whether something satifies our criteria for being a model, a language or a transformation) and minimality (it is hard to imagine a case of modelling or transformation not having the characterstics that we propose).

Keywords: Model, language, transformation

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/24

MDA-E: Model Drive Architecture Evolution

Bernhard Rumpe (TU Braunschweig)

Building software and software-based systems becomes an increasingly complex task. Business systems, e.g., started out as stand-alone applications, then where integrated within whole companies and currently become connected through the internet in e-commerce platforms that taken into its extreme finally builds a single world-wide business software system.

Driven by complexity and diversification, a portfolio of development techniques that are applicable in a variety of development contexts has emerged. Such a portfolio allows developers to select and adapt processes, methods, and techniques based on the application domain and the criticality and complexity of the system to be built. New application domains, such as e-commerce, require flexible approaches to meeting the to develop high quality software cheaply and rapidly. This talk describes an overview of work on integrating two of those development approaches. UMLbased approaches on the one hand emphasize the modeling of software from a variety of views. Agile

development methods on the other hand de-emphasize software modeling, while emphasizing disciplined coding and testing. The integrated approach is based on the use of an adapted version of the UML as a modelling, coding and test notation. The talk examines how the UML needs to be adapted to support rapid development and how the software development process stands to benefit from the adaptation. In particular the evolution of the architecture described by UML models according to changing requirements or technology through systematic adaptation ("refactoring") steps will be discussed, its underlying theory explored, and demonstrated using examples. The talk will also address the relationship between refactoring of architectural models and code generation, automated tests and common ownership of models.

See also: B. Rumpe: Agile Modellierung mit der UML, Springer 2004

Graph Transformation Based Models of Dynamic Software Architectures and Architectural Styles

Sebastian Thöne (University of Paderborn)

Software architectures play an important role in software development. As abstract models of the run-time structure they help to bridge the gap between user requirements and implementation. In the context of e-business, self-healing, or mobile systems, *dynamic architectures* gain more and more importance. They represent systems that do not simply consist of a fixed, static structure, but can react to certain requirements or events by run-time reconfiguration of its components and connections. The availability of those reconfiguration operations depends on the chosen run-time platform which has to support the desired modifications.

The development of such dynamic architectures is a complex task which is usually driven by a stepwise modeling and refinement approach. The software architect derives a first abstract model of the architecture from the user requirements. This model mainly covers the functional aspects and business-related components. Later in the design process, more and more non-functional requirements like security concepts and implementation-specific aspects are integrated into the core functionality. This leads to a sequence of refined architectures down to the real system design for implementation.

A recent example of this general modeling principle is the *Model-Driven Architecture (MDA)* put forward by the OMG. Here, platform-specific details are initially ignored at the model-level to allow for maximum portability. Then, these platform-independent models are refined by adding details required to map to a given target platform. Thus, at each refinement level, one imposes more assumptions on the resources, constraints, and services of the chosen platform. In software architecture research, *architectural styles* are used to describe families of architectures by common resource types, configuration patterns and constraints. We propose in [1] to consider the restrictions imposed by a certain choice of platform as an architectural style. Moreover, to account for component interactions and platforms that support dynamic reconfigurations, we extend the classical notion of architectural style, which is restricted to structural constraints, by also describing platform-specific communication and reconfiguration mechanisms.

We formalize the architectural styles as graph transformation systems including architectural types, constraints, and graph transformation rules. Based on that, a notion of *refinement* is defined in [2], which preserves both semantic correctness and platform

consistency. This means that a concrete architecture must satisfy the same requirements as the abstract architecture, and that it must be consistent with constraints and mechanisms imposed by the chosen target platform.

For this purpose, we define refinement relations between abstract and concrete styles which enable us to check for correct refinement of two given architectures. We do not only consider *structural* refinements of fixed configurations but also *behavioral* refinement, which means refining abstract scenarios of component interactions and reconfigurations into platform-specific scenarios. Since refinements are often tedious and error-prone, a further goal of our work is to automate the derivation of refined models. Indeed, the maximum gain of reusing platform-independent models is achieved if the mapping to various target platforms can be automated. For this purpose, we propose a formulation of the behavioral refinement problem as a reachability problem which can be solved by classical graph transformation and model checking tools.

Joint work of: Sebastian Thöne, Luciano Baresi, Reiko Heckel, and Dániel Varró

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2005/17

References:

[1] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Modeling and validation of serviceoriented architectures: Application vs. style. In Proc. ESEC/FSE 03 European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 68 77. ACM Press, 2003. http://wwwcs.upb.de/cs/agengels/Papers/2003/BaresiHeckelThoeneVarro_ESEC03.pdf.

[2] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Style-based refinement of dynamic software architectures. In Proc. WICSA4 4^{th} Working IEEE/IFIP Conference on Software Architecture, 2004. to appear. http://wwwcs.upb.de/cs/agengels/Papers/2004/WICSA4_Baresi-Heckel-Thoene-Varro.pdf.

Does one size of model transformation language fit all?

Laurence Tratt (King's College - London)

Much of the recent focus on model transformations has been on finding "the" model transformation language - a single language that performs all sorts of model transformations well. Debates have raged over different approaches, and yet we do not appear to be significantly closer to finding a single model transformation language which is likely to cope with different organizations criteria. In this talk I will present an overview of some of the major approaches to model transformations, explain where different approaches can learn from one another, and finally present some thoughts on the applicability of different approaches to different situations. Are we looking at a "two speed" future for model transformations or is there a size which fits all?

Write Once, Deploy N: a Performance Oriented MDA Case Study

Pieter Van Gorp (University of Antwerpen)

To focus the comparison of languages for model checking and transformation on criteria that matter in practical development, there is an urgent need for more, realistic case studies. In this paper, we first present the problem of developing distributed database applications that are optimized for concurrent data access, without locking in on vendor extensions of a particular J2EE application server, with proper separation of concerns, and with tool support for domain evolution. Then, we propose and discuss a conceptual language for model refinement and code generation as a possible solution to the presented problem. After applying this particular language on our case study, we derive general conclusions on composition, sequencing, inheritance, and design by contract for such languages.

Keywords: Model transformation, consistency management, qvt, mda, ocl

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/18

Towards Automated Formal Verification of Modelling Languages by Model Checking Techniques

Daniel Varro (Budapest University of Technology and Economics)

I present a method with tool support for model checking dynamic consistency properties in arbitrary well-formed instance models of any modeling language defined visually by metamodeling and graph transformation techniques. Our tool (CheckVML) first translates such high-level specifications into a tool independent abstract representation of transition systems defined by a corresponding metamodel. From this intermediate representation the input language of the back-end model checker tool (i.e., SPIN in our case) is generated automatically.

Keywords: Graph transformation, model checking, metamodeling,

Refinement and Consistency in Multiview Models

Heike Wehrheim (Universität Oldenburg)

Model transformations are an integral part of OMG's stan- dard for Model Driven Architecture (MDA). Model transformations should at the best allow for a seamless transition from high-level models to ac- tual implementations. They are therefore required to be behaviour pre- serving: models (or the final implementation) at lower levels should ad- here to the descriptions given in higher level models. Moreover, for com- plex systems models usually consists of descriptions of different views on the system. Consequently, different kinds of model transformations take place on different views, and together they should guarantee behaviour- preservation. In this paper we discuss the applicability of formal methods to model transformations. Formal methods

come with build-in notions of trans- formations between models, or more precisely, with refinement and sub- typing concepts which provide means for comparing models on different levels with respect to their behaviour. Such notions can be applied as correctness criteria for evaluating model transformations. Moreover, re- finement and subtyping concepts for different views can be shown to neatly fit together. This is achieved by giving a common semantics to all views which furthermore opens the possibility of checking consistency between them.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2005/19

A Program Design Language for Mobile Architectures

Michel Wermelinger (Universidade Nova de Lisboa)

Mobility, as enabled by the internet and wireless communication, is introducing an additional factor of complexity in software development. Methods and techniques have to account for the changes that can occur, at run time, at the level of the topology over which components perform computations and interact with one another.

Architectural modelling techniques have helped to tame the complexity of building distributed applications over static networks. This is because they enforce a strict separation of concerns between what in systems can account for the "computations" that are responsible for the behaviour that individual components ensure locally, and the mechanisms that control their behaviour and coordinate the interconnections through which global properties of systems can emerge. As a consequence, one can build complex systems from simpler components by superposing the architectural connectors that coordinate their interactions.

A major effort is being pursued within the IST-2001-32747 project *AGILE – Architectures for Mobility* – to take this separation of concerns one step further and address distribution/mobility aspects as a first-class architectural dimension. More precisely, we are extending a prototype program design language – *CommUnity* – that we have used for experimenting with architectural design methods and techniques [2,6], with primitives that support the construction and evolution of location-aware architectural models by superposing explicit connectors that handle mobility aspects.

Information on this extension is already available in several publications. In [5] we can find an early introduction and motivation for the new primitives, which is refined and extended in [4] with a more explicit separation of concerns based on distribution connectors and support for compositional and incremental refinement. An early experiment is reported in [1] based on the airport luggage delivery system. A more sophisticated account of this case study can be found in [4]. Updated information can be found at http://www.fiadeiro.org/jose/CommUnity.

References:

1. L.F.Andrade, J.L.Fiadeiro, A.Lopes and M.Wermelinger, "Architectural Techniques for Evolving Control Systems", in Formal Methods for Railway Operation and Control Systems, G.Tarnai and E.Schnieder (eds), L'Harmattan Press 2003
2. J.L.Fiadeiro, A.Lopes and M.Wermelinger, "A Mathematical Semantics for Architectural Connectors", in Generic Programming, R.Backhouse and J.Gibbons (eds), LNCS 2793, 190-234, Springer-Verlag 2003. 2003

3. J.L.Fiadeiro and A.Lopes, "CommUnity on the Move: Architectures for Distribution and Mobility", in Formal Methods for Components and Objects, F.deBoer and M.Bonsague (eds), Springer-Verlag 2004.
4. A.Lopes and J.L.Fiadeiro, "Adding Mobility to Software Architectures", A.Brogi and J.- M.Jacquet (eds), FOCLASA 2003 Ű Foundations of Coordination Languages and Software Architecture, Electronic Notes in Theoretical Computer Science. Elsevier Science, in print.
5. A.Lopes, J.L.Fiadeiro and M.Wermelinger, "Architectural Primitives for Distribution and Mobility", in Proc. SIGSOFT 2002/FSE-10, 41-50, ACM Press, 2002.
6. M.Wermelinger and J.Fiadeiro, "Connectors for Mobile Programs", IEEE Transactions on Software Engineering 24(5), 331-341, 1998.

Joint work of: Wermelinger, Michel; Lopes, Antónia; Fiadeiro, José

See also: http://www.fiadeiro.org/jose/CommUnity